autogluon_all

October 23, 2023

1 Config

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*10
presets = "best_quality"#'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_outliers = True

to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
 ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
 ↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
 ↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
 ↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
 ↪"pressure_100m:hPa"]

#to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:
 ↪ms",␣"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:
 ↪cm",␣"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:
 ↪mm",␣"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm",␣
 ↪"absolute_humidity_2m:gm3", "air_density_2m:kgm3"]

use_groups = False
n_groups = 8

auto_stack = True
num_stack_levels = None# 1
num_bag_folds = None# 8
num_bag_sets = None#20
```

```
use_tune_data = True
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[2]:
```
import pandas as pd
import numpy as np



import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()

    # Create a set for constant-time lookup
```

```python
    index_set = set(X.index)

    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
↪loc[shifted_indices[shifted_indices.isin(index_set)]][columns]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    date_calc = None
    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']



    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X



def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
```

```python
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600
```

```
        X_train_estimated["estimated_diff_hours"] =␣
↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1


    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    print(f"Size of estimated after dropping nans:␣
↪{len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
```

```python
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
```

```
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954
```

2.1 Feature enginering

2.1.1 Remove anomalies

[3]:
```python
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}

        for idx in x.index:
            value = x[idx]
```

```
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

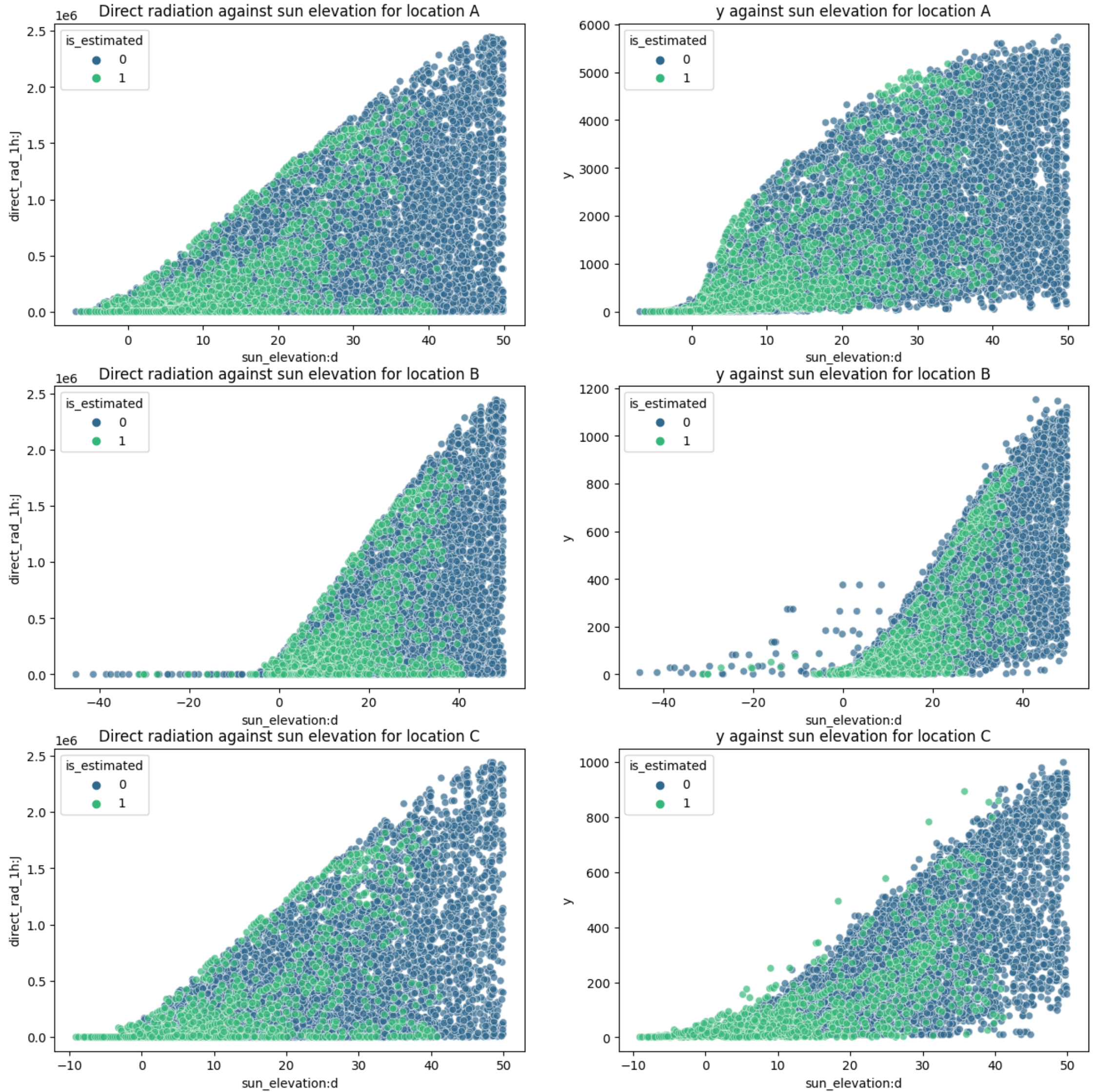
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[4]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

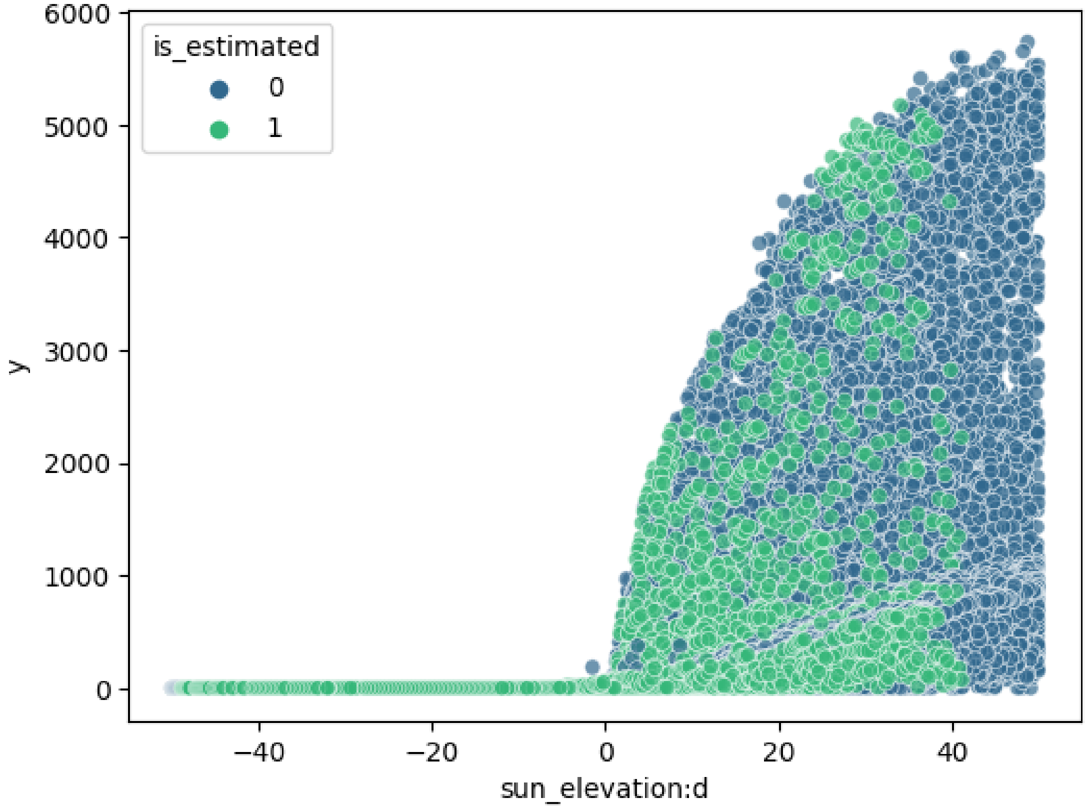
```
Dropped rows:  9293
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]
```

```python
# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for␣
 ↪location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```

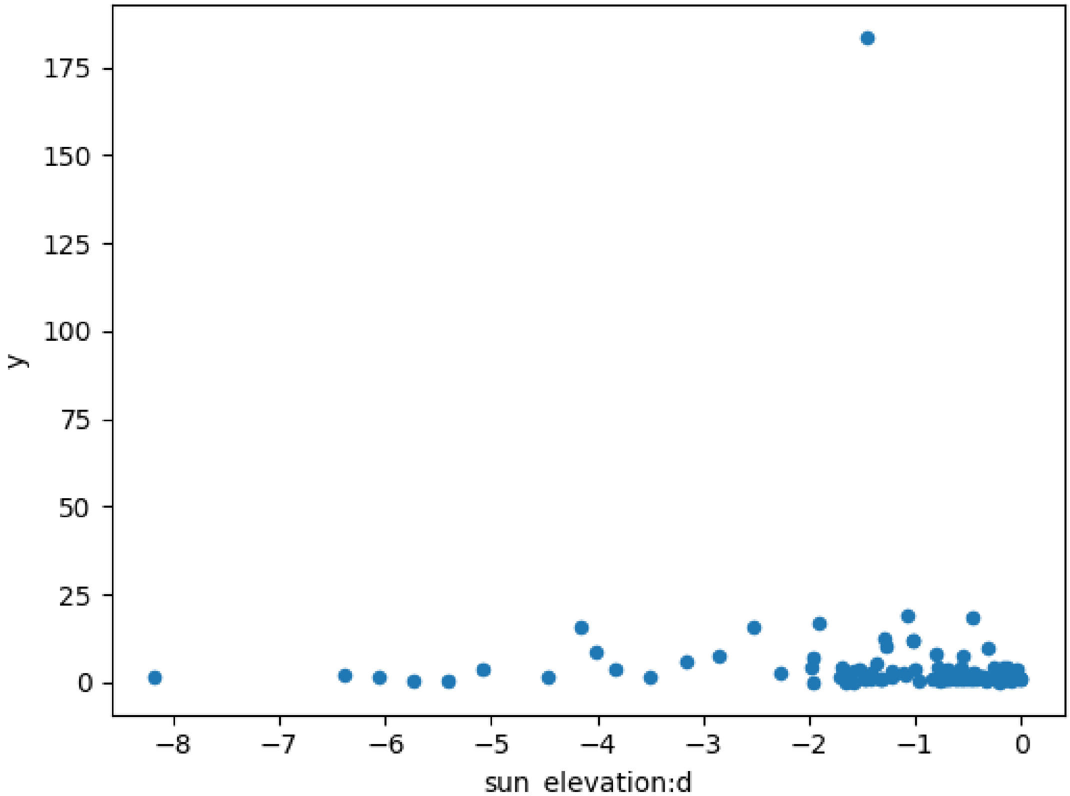


[6]:
```python
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=
    thresh) & (X_train["y"] >= 0.1)
if drop_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",
    palette="viridis", alpha=0.7)
plt.show()
```



[7]:
```
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

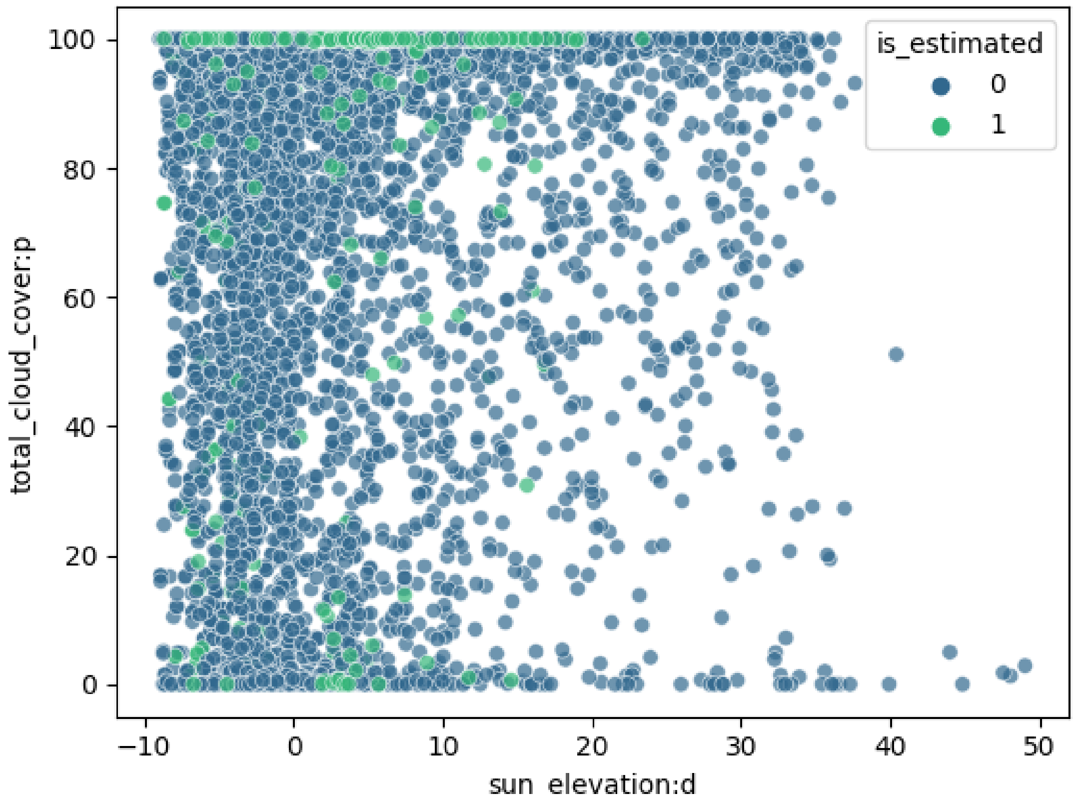
[7]: <AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>



[8]:
```
# set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0␣
↪(or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.001
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.001
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |␣
↪(X_train["diffuse_rad_1h:J"] > threshold_diffuse))
print(len(X_train[mask]))

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="total_cloud_cover:
↪p", hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",␣
↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)
```

```python
# set y to nan where mask
if drop_outliers:
    X_train.loc[mask, "y"] = np.nan

# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]
```

```
Threshold direct: 2445.897
Threshold diffuse: 1182.2505
7864
```



[8]:
```
location  is_estimated
A         0               1398
          1                194
B         0               2977
          1                213
C         0               2927
          1                155
```

```
Name: direct_rad_1h:J, dtype: int64
```

[9]:
```python
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  9740
```

2.1.2 Other stuff

[10]:
```python
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,
 ↪multiply sample_weight with sample_weight_may_july if the ds is between
 ↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to
 ↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=
 ↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))


if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location
```

```
    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())
```

```
X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3                7.025
air_density_2m:kgm3                     1.2245
ceiling_height_agl:m                 3423.625
clear_sky_energy_1h:J             1028475.5
clear_sky_rad:W                    179.350006
cloud_base_agl:m                     3423.625
dew_or_rime:idx                         0.0
dew_point_2m:K                        279.125
diffuse_rad:W                       38.674999
diffuse_rad_1h:J                     259207.0
direct_rad:W                            9.35
direct_rad_1h:J                  132030.90625
effective_cloud_cover:p             83.974998
elevation:m                             6.0
fresh_snow_12h:cm                       0.0
```

```
fresh_snow_1h:cm                              0.0
fresh_snow_24h:cm                             0.0
fresh_snow_3h:cm                              0.0
fresh_snow_6h:cm                              0.0
is_day:idx                                    1.0
is_in_shadow:idx                              0.0
msl_pressure:hPa                      1019.200012
precip_5min:mm                                0.0
precip_type_5min:idx                          0.0
pressure_100m:hPa                     1006.450012
pressure_50m:hPa                      1012.450012
prob_rime:p                                   0.0
rain_water:kgm2                               0.0
relative_humidity_1000hPa:p             49.424999
sfc_pressure:hPa                      1018.474976
snow_density:kgm3                             NaN
snow_depth:cm                                 0.0
snow_drift:idx                                0.0
snow_melt_10min:mm                            0.0
snow_water:kgm2                               0.0
sun_azimuth:d                          294.874268
sun_elevation:d                            13.592
super_cooled_liquid_water:kgm2                0.2
t_1000hPa:K                            287.049988
total_cloud_cover:p                        85.125
visibility:m                         50129.550781
wind_speed_10m:ms                             4.5
wind_speed_u_10m:ms                          -4.2
wind_speed_v_10m:ms                         1.475
wind_speed_w_1000hPa:ms                       0.0
is_estimated                                    0
y                                           45.98
location                                        A
Name: 2019-06-12 18:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00                       7.7               1.2235

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 23:00:00           1689.824951                    0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00              0.0       1689.824951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
```

```
2019-06-02 23:00:00      280.299988           0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00   286.899994                100.0  33770.648438

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00               3.35                -3.35

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00                0.275                      0.0

                     is_estimated    y  location
ds
2019-06-02 23:00:00             0  0.0         A

[1 rows x 48 columns]
```

[11]:
```
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
    plt.show()
```

[12]:
```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
```

```
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)

    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[13]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 2500.0/3
    if use_test_data:
        tune_rows = max(3125.0/3, len(estimated_set[estimated_set["location"]␣
↪== location]))

    holdout_frac = max(0.01, min(0.04, tune_rows / num_train_rows)) *␣
↪num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}:␣
↪{len(estimated_set[estimated_set['location'] == location])}. Holdout frac␣
↪should be % of estimated: {holdout_frac}")

    # shuffle and split data
```

```
    loc_tune_set, loc_new_train_set =␣
↪split_and_shuffle_data(estimated_set[estimated_set['location'] == location],␣
↪40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.
↪2)
        test_set = pd.concat([test_set, loc_test_set])

    tune_set = pd.concat([tune_set, loc_tune_set])




print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())




if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])


train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
↪train data.
```

```
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Size of estimated for location A: 4020. Holdout frac should be % of estimated:
0.3103283582089552
Length of location tune set : 1246
Size of estimated for location B: 3320. Holdout frac should be % of estimated:
0.31365060240963855
Length of location tune set : 1040
Size of estimated for location C: 2768. Holdout frac should be % of estimated:
0.3299421965317919
Length of location tune set : 882
Length of train set before adding test set 69945
Length of train set after adding test set 76885
Shapes of tuning data location
A    1005
B     840
C     722
dtype: int64
Shape of test 601
```

3 Quick EDA

[14]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,
 ↪label="y", sample=None)
```

[15]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[16]:
```
import os


# Get the last submission number
```

```
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 108
Now creating submission number: 109
New filename: submission_109
```

[17]:
```
def fit_predictor_for_location():
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:", train_data["sample_weight"].
↪sum())
        print("Train data number of rows:", train_data.shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:", tuning_data["sample_weight"].
↪sum())
            print("Tune data number of rows:", tuning_data.shape[0])
        if use_test_data:
            print("Test data sample weight sum:", test_data["sample_weight"].
↪sum())
            print("Test data number of rows:", test_data.shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_all",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data.drop(columns=["ds"]),
        time_limit=time_limit,
        presets=presets,
        # num_stack_levels=num_stack_levels,
        # num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
```

```
        # num_bag_sets=num_bag_sets,
        tuning_data=tuning_data.reset_index(drop=True).drop(columns=["ds"]) if␣
↪use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        perf = predictor.evaluate(test_data)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

predictor = fit_predictor_for_location()
predictors = [predictor, predictor, predictor]
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_109_all"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_109_all/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.191-1 (2023-08-16)
Disk Space Avail:   286.43 GB / 315.93 GB (90.7%)
Train Data Rows:    76885
Train Data Columns: 32
Tuning Data Rows:    2567
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, -0.0, 334.51249,
826.0705)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129981.51 MB
```

```
        Train Data (Original)  Memory Usage: 24.31 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
                Fitting CategoryFeatureGenerator…
                        Fitting CategoryMemoryMinimizeFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Types of features in original data (raw dtype, special dtypes):
                ('float', [])  : 30 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])    :  1 | ['is_estimated']
                ('object', []) :  1 | ['location']
        Types of features in processed data (raw dtype, special dtypes):
                ('category', [])  :  1 | ['location']
                ('float', [])     : 30 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.3s = Fit runtime
        32 features in original data used to generate 32 features in processed
data.
        Train Data (Processed) Memory Usage: 19.23 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.36s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
```

```
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 399.66s of the
599.64s of remaining time.
        -210.466         = Validation score   (-mean_absolute_error)
        0.07s    = Training   runtime
        1.64s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 397.75s of the
597.72s of remaining time.
        -252.7209        = Validation score   (-mean_absolute_error)
        0.07s    = Training   runtime
        1.65s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 395.86s of the
595.84s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -46.2736         = Validation score   (-mean_absolute_error)
        46.6s    = Training   runtime
        56.48s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 334.9s of the
534.88s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -48.486  = Validation score   (-mean_absolute_error)
        45.67s   = Training   runtime
        53.51s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 278.84s of
the 478.81s of remaining time.
        -54.1401         = Validation score   (-mean_absolute_error)
        16.01s   = Training   runtime
        2.62s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 258.44s of the
458.42s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
        -54.9723         = Validation score   (-mean_absolute_error)
        206.94s  = Training   runtime
        0.26s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 50.28s of the
250.26s of remaining time.
        -54.5672         = Validation score   (-mean_absolute_error)
        3.2s     = Training   runtime
        2.59s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 42.69s of the
242.67s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -60.4248         = Validation score   (-mean_absolute_error)
        32.98s   = Training   runtime
        1.23s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 6.75s of the 206.73s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -53.7168         = Validation score   (-mean_absolute_error)
        5.51s    = Training   runtime
        0.63s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
199.13s of remaining time.
        -45.6788         = Validation score   (-mean_absolute_error)
        0.39s    = Training   runtime
        0.0s     = Validation runtime
Fitting 9 L2 models …
Fitting model: LightGBMXT_BAG_L2 … Training model for up to 198.73s of the
198.71s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -39.5863         = Validation score   (-mean_absolute_error)
        34.8s    = Training   runtime
        5.15s    = Validation runtime
Fitting model: LightGBM_BAG_L2 … Training model for up to 159.58s of the
159.56s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -40.6695         = Validation score   (-mean_absolute_error)
        3.95s    = Training   runtime
        0.33s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L2 … Training model for up to 154.07s of
the 154.05s of remaining time.
        -40.0341         = Validation score   (-mean_absolute_error)
        26.93s   = Training   runtime
```

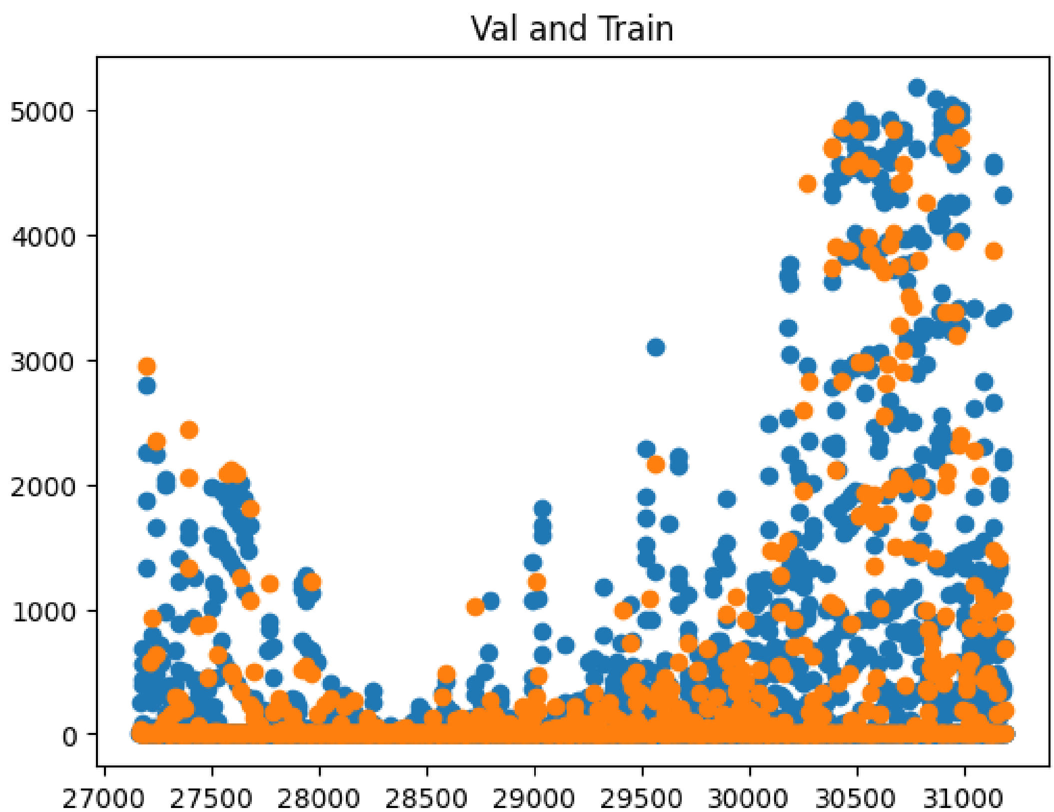
```
        2.98s    = Validation runtime
Fitting model: CatBoost_BAG_L2 … Training model for up to 122.28s of the
122.26s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -41.6149         = Validation score   (-mean_absolute_error)
        42.79s   = Training   runtime
        0.11s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 … Training model for up to 78.15s of the
78.13s of remaining time.
        -41.4044         = Validation score   (-mean_absolute_error)
        4.43s    = Training   runtime
        2.94s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 … Training model for up to 68.91s of the
68.89s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -42.8463         = Validation score   (-mean_absolute_error)
        56.14s   = Training   runtime
        1.39s    = Validation runtime
Fitting model: XGBoost_BAG_L2 … Training model for up to 10.68s of the 10.66s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -40.3126         = Validation score   (-mean_absolute_error)
        4.13s    = Training   runtime
        0.4s     = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 … Training model for up to 5.1s of the
5.08s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        Time limit exceeded… Skipping NeuralNetTorch_BAG_L2.
Fitting model: LightGBMLarge_BAG_L2 … Training model for up to 1.36s of the
1.34s of remaining time.
2023-10-23 11:55:33,796 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,798 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,800 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,802 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,803 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
```

```
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,804 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
2023-10-23 11:55:33,805 ERROR worker.py:399 -- Unhandled error (suppress with
'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly while executing
this task. Check python-core-worker-*.log files for more information.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -121.4584        = Validation score   (-mean_absolute_error)
        1.73s    = Training   runtime
        0.11s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L3 … Training model for up to 360.0s of the
-1.86s of remaining time.
        -39.2779         = Validation score   (-mean_absolute_error)
        0.33s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 602.24s … Best model:
"WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_109_all/")
Evaluation: mean_absolute_error on test data: -36.992554823492156
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -36.992554823492156,
    "root_mean_squared_error": -116.93409921366413,
    "mean_squared_error": -13673.583558911045,
    "r2": 0.9512652500258204,
    "pearsonr": 0.9753306005016983,
    "median_absolute_error": -0.3539777100086212
}

Evaluation on test data:
-36.992554823492156
```

[18]:

```python
import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &␣
↪(train_data["is_estimated"]==True)]["y"].index,␣
↪train_data[(train_data["location"] == loc) &␣
↪(train_data["is_estimated"]==True)]["y"])
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
↪tuning_data[tuning_data["location"] == loc]["y"])
```

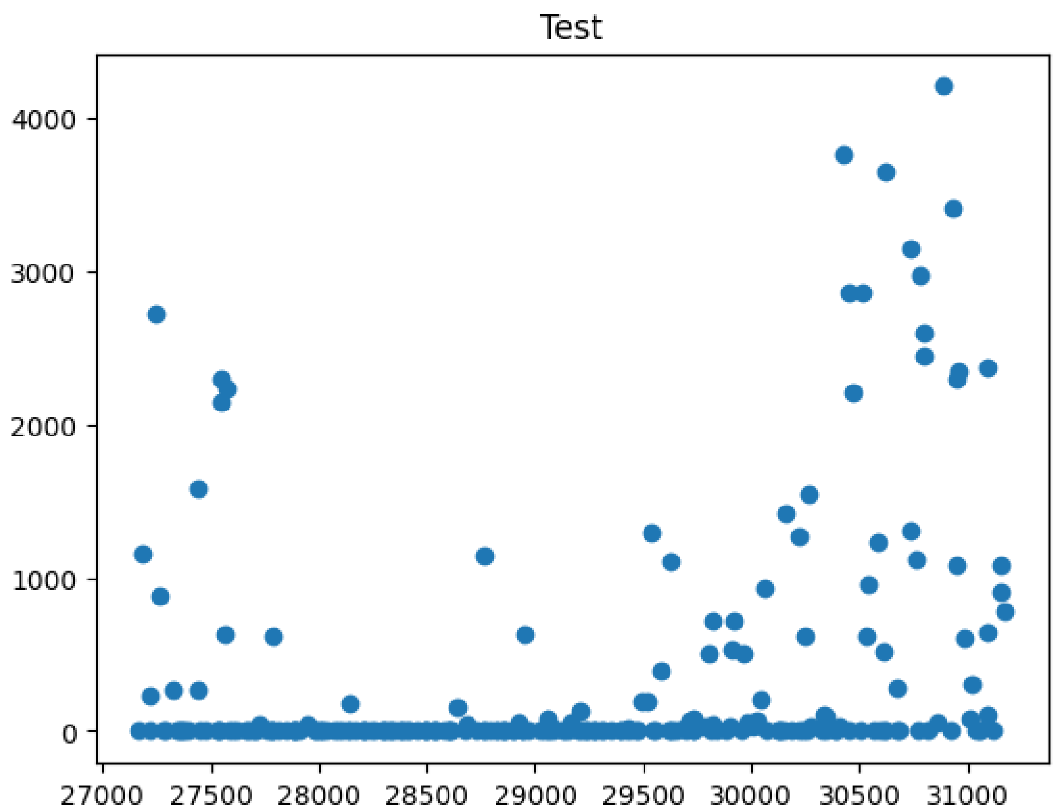
```
        plt.title("Val and Train")
        plt.show()

        if use_test_data:
            lb = predictors[i].leaderboard(test_data[test_data["location"] ==␣
 ↪loc])
            lb["location"] = loc
            plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
 ↪test_data[test_data["location"] == loc]["y"])
            plt.title("Test")

            return lb

    return pd.DataFrame()
loc = "A"
leaderboards[0] = leaderboard_for_location(0, loc)
```




```
                        model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0      WeightedEnsemble_L3  -72.825492  -39.277911        5.183208
```

```
130.127033  475.250387                 0.003186                0.000696
0.329261          3       True          19
1   RandomForestMSE_BAG_L2  -73.061870  -40.034063         4.638613
123.587410  383.981505                 0.832793                2.983340
26.928309          2       True          13
2           CatBoost_BAG_L2  -73.713897  -41.614909         3.838225
120.715683  399.842792                 0.032404                0.111613
42.789596          2       True          14
3         LightGBMXT_BAG_L2  -73.924817  -39.586332         4.148858
125.755889  391.850575                 0.343037                5.151819
34.797379          2       True          11
4           LightGBM_BAG_L2  -75.140145  -40.669485         3.868616
120.936924  360.998679                 0.062795                0.332854
3.945484          2       True          12
5            XGBoost_BAG_L2  -75.176360  -40.312638         3.890949
121.002223  361.179377                 0.085128                0.398152
4.126181          2       True          17
6   NeuralNetFastAI_BAG_L2  -75.441927  -42.846275         4.004192
121.991178  413.195437                 0.198371                1.387108
56.142241          2       True          16
7      ExtraTreesMSE_BAG_L2  -76.923756  -41.404359         4.666742
123.547206  361.481033                 0.860922                2.943136
4.427837          2       True          15
8       WeightedEnsemble_L2  -77.042957  -45.678779         2.782619
113.829946  141.643967                 0.003736                0.000805
0.389164          2       True          10
9         LightGBMXT_BAG_L1  -78.178231  -46.273607         0.934425
56.480547   46.595445                 0.934425               56.480547
46.595445          1       True           3
10          LightGBM_BAG_L1  -81.266506  -48.485993         0.871377
53.506534   45.674402                 0.871377               53.506534
45.674402          1       True           4
11           XGBoost_BAG_L1  -92.993775  -53.716756         0.121918
0.630693    5.508166                 0.121918                0.630693
5.508166          1       True           9
12  RandomForestMSE_BAG_L1  -93.033273  -54.140131         0.805297
2.616562   16.006378                 0.805297                2.616562
16.006378          1       True           5
13          CatBoost_BAG_L1  -94.232793  -54.972334         0.050310
0.262134  206.944024                 0.050310                0.262134
206.944024          1       True           6
14     ExtraTreesMSE_BAG_L1  -95.454539  -54.567232         0.809866
2.593719    3.202688                 0.809866                2.593719
3.202688          1       True           7
15  NeuralNetFastAI_BAG_L1 -101.052613  -60.424840         0.167783
1.225498   32.978578                 0.167783                1.225498
32.978578          1       True           8
16    LightGBMLarge_BAG_L2 -169.590028 -121.458419         3.900330
```

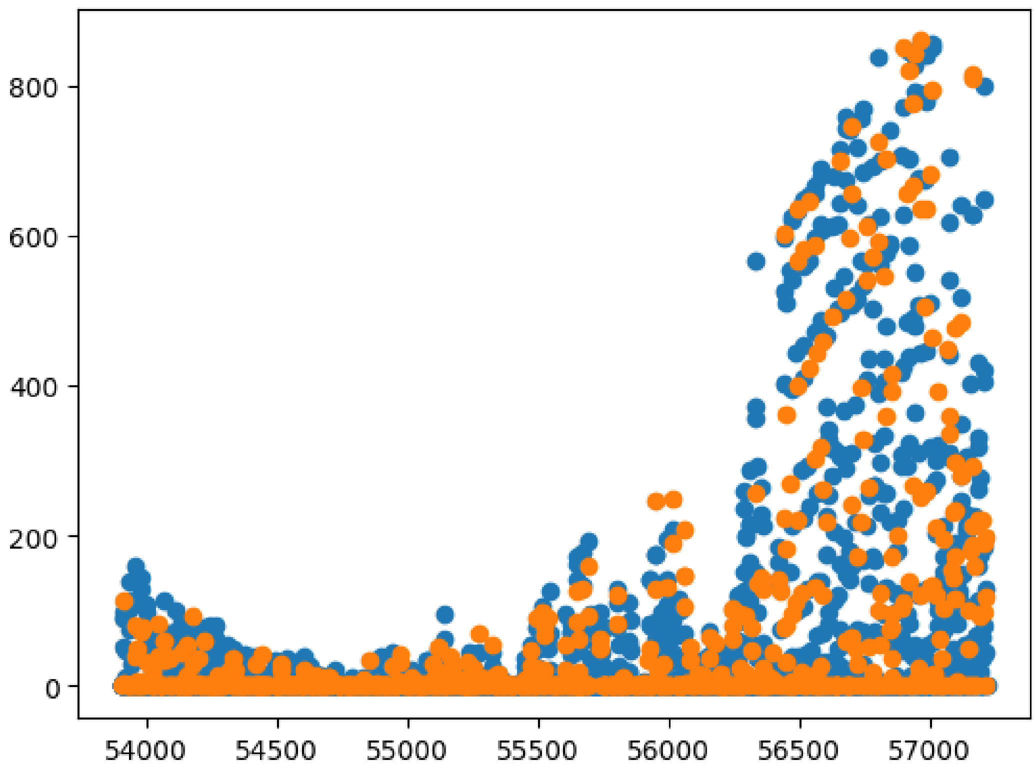
```
120.715600  358.779523                   0.094509                 0.111530
1.726327            2        True          18
17   KNeighborsUnif_BAG_L1 -240.695407 -210.466014        0.026368
1.640172    0.072643                   0.026368                 1.640172
0.072643            1        True           1
18   KNeighborsDist_BAG_L1 -271.502625 -252.720925        0.018476
1.648211    0.070871                   0.018476                 1.648211
0.070871            1        True           2
```



[20]:

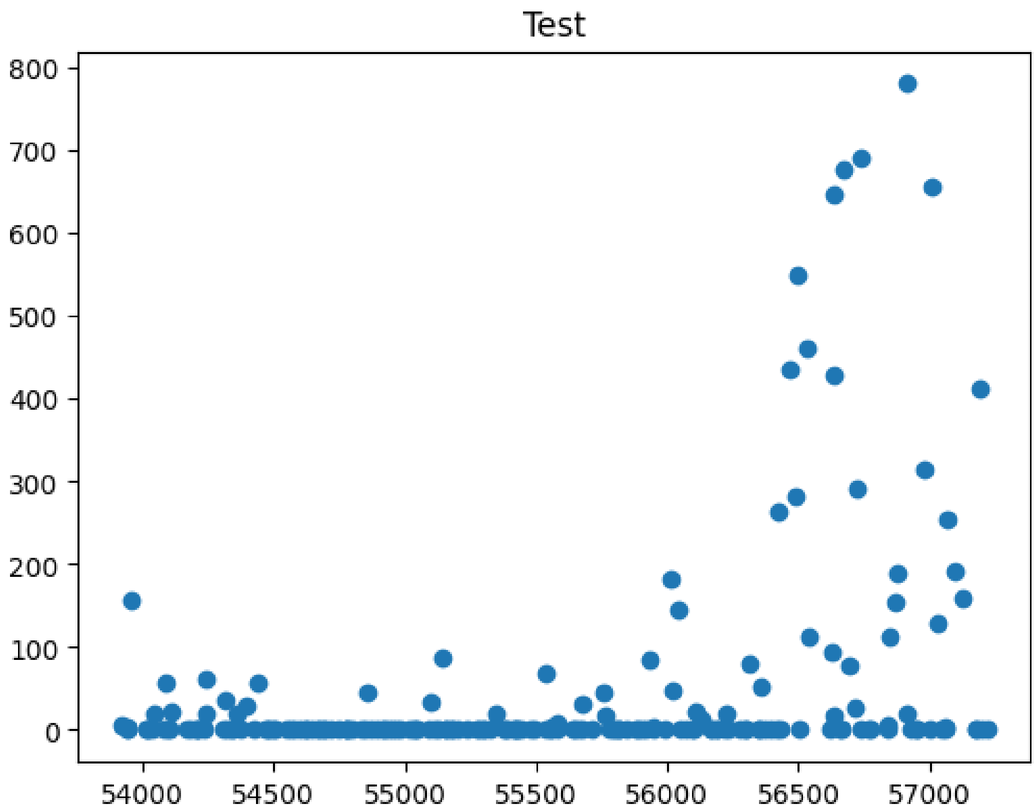
```
loc = "B"
leaderboards[1] = leaderboard_for_location(1, loc)
```

Val and Train

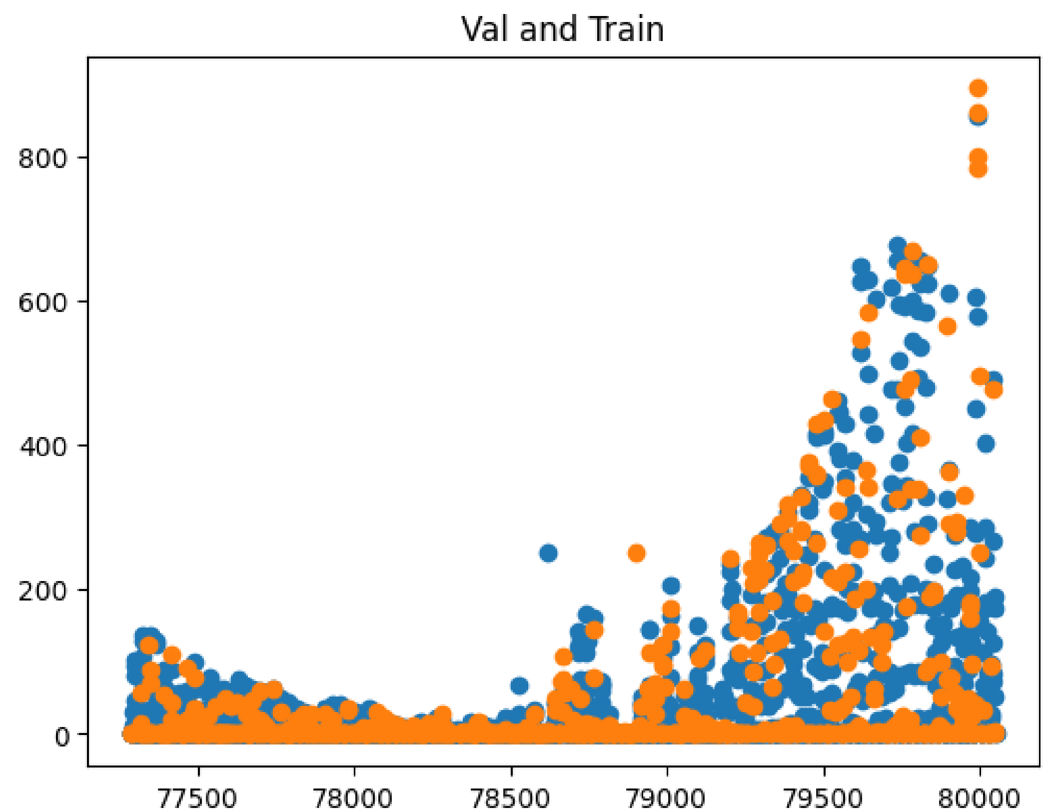



```
                        model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0      ExtraTreesMSE_BAG_L2  -10.525107  -41.404359        4.584899
123.547206  361.481033                 0.819914               2.943136
4.427837            2        True         15
1    RandomForestMSE_BAG_L2  -11.013135  -40.034063        4.590575
123.587410  383.981505                 0.825591               2.983340
26.928309            2        True         13
2       WeightedEnsemble_L3  -11.216250  -39.277911        5.153603
130.127033  475.250387                 0.003202               0.000696
0.329261            3        True         19
3          LightGBMXT_BAG_L2  -11.733009  -39.586332        4.124192
125.755889  391.850575                 0.359208               5.151819
34.797379            2        True         11
4             XGBoost_BAG_L2  -11.986381  -40.312638        3.848172
121.002223  361.179377                 0.083188               0.398152
4.126181            2        True         17
5           LightGBM_BAG_L2  -12.553477  -40.669485        3.824061
120.936924  360.998679                 0.059076               0.332854
3.945484            2        True         12
```

```
6          CatBoost_BAG_L2  -12.685649  -41.614909        3.793782
120.715683  399.842792                 0.028797                 0.111613
42.789596            2       True         14
7      WeightedEnsemble_L2  -13.776032  -45.678779        2.751264
113.829946  141.643967                 0.003772                 0.000805
0.389164            2       True         10
8   RandomForestMSE_BAG_L1  -13.853536  -54.140131        0.792803
2.616562   16.006378                 0.792803                 2.616562
16.006378            1       True          5
9     ExtraTreesMSE_BAG_L1  -14.983459  -54.567232        0.819206
2.593719    3.202688                 0.819206                 2.593719
3.202688            1       True          7
10  NeuralNetFastAI_BAG_L2  -15.297022  -42.846275        3.965602
121.991178  413.195437                 0.200617                 1.387108
56.142241            2       True         16
11          XGBoost_BAG_L1  -15.388737  -53.716756        0.100228
0.630693    5.508166                 0.100228                 0.630693
5.508166            1       True          9
12         CatBoost_BAG_L1  -15.800668  -54.972334        0.050005
0.262134  206.944024                 0.050005                 0.262134
206.944024            1       True          6
13       LightGBMXT_BAG_L1  -15.912962  -46.273607        0.915757
56.480547   46.595445                 0.915757                56.480547
46.595445            1       True          3
14         LightGBM_BAG_L1  -18.537402  -48.485993        0.876585
53.506534   45.674402                 0.876585                53.506534
45.674402            1       True          4
15  NeuralNetFastAI_BAG_L1  -20.884978  -60.424840        0.162347
1.225498   32.978578                 0.162347                 1.225498
32.978578            1       True          8
16     LightGBMLarge_BAG_L2  -84.468628 -121.458419        3.832304
120.715600  358.779523                 0.067319                 0.111530
1.726327            2       True         18
17   KNeighborsUnif_BAG_L1 -130.994529 -210.466014        0.013962
1.640172    0.072643                 0.013962                 1.640172
0.072643            1       True          1
18   KNeighborsDist_BAG_L1 -250.579001 -252.720925        0.034092
1.648211    0.070871                 0.034092                 1.648211
0.070871            1       True          2
```

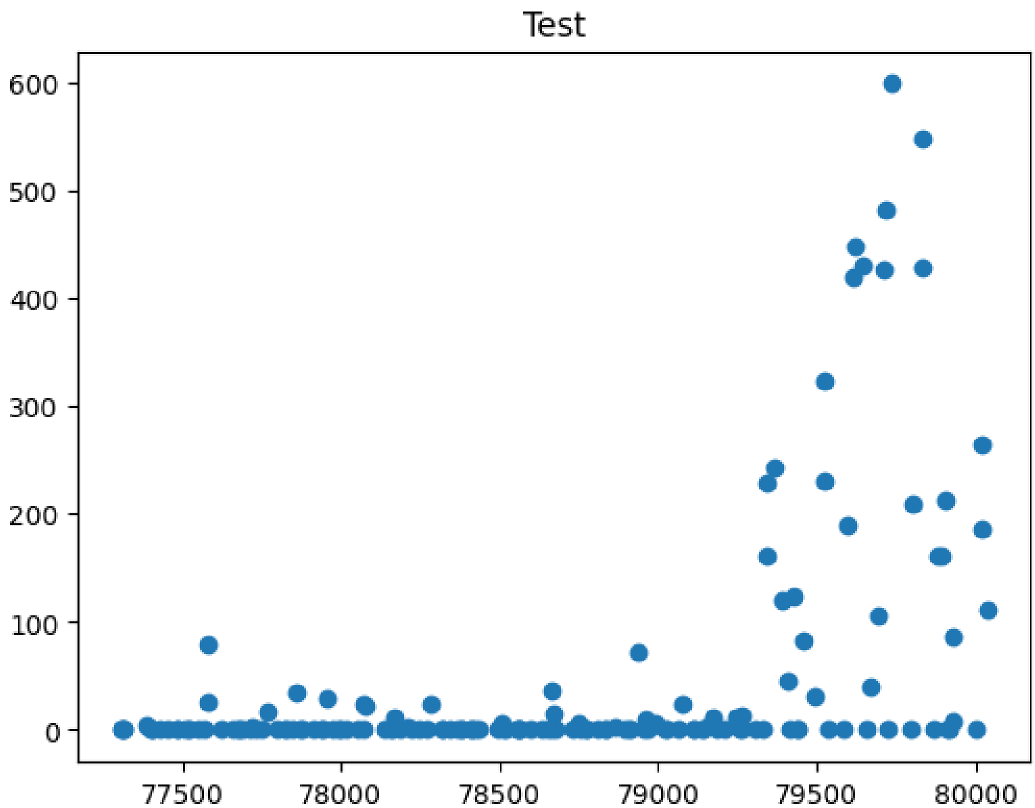


[21]:
```
loc = "C"
leaderboards[2] = leaderboard_for_location(2, loc)
```




```
                          model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0          LightGBMXT_BAG_L2  -14.819934  -39.586332        4.776722
125.755889  391.850575                 0.618163             5.151819
34.797379            2        True          11
1       ExtraTreesMSE_BAG_L2  -14.873456  -41.404359        4.983536
123.547206  361.481033                 0.824977             2.943136
4.427837            2        True          15
2       ExtraTreesMSE_BAG_L1  -14.926046  -54.567232        0.822037
2.593719     3.202688                0.822037             2.593719
3.202688            1        True           7
3        WeightedEnsemble_L3  -15.239574  -39.277911        5.800717
130.127033  475.250387                 0.003144             0.000696
0.329261            3        True          19
4             XGBoost_BAG_L2  -15.631105  -40.312638        4.240143
121.002223  361.179377                 0.081583             0.398152
4.126181            2        True          17
5    RandomForestMSE_BAG_L2  -15.758504  -40.034063        4.987351
123.587410  383.981505                 0.828792             2.983340
26.928309            2        True          13
```

```
6          LightGBM_BAG_L2  -16.000889  -40.669485        4.444362
120.936924  360.998679                 0.285803                0.332854
3.945484           2       True         12
7          CatBoost_BAG_L2  -16.441365  -41.614909        4.187279
120.715683  399.842792                 0.028720                0.111613
42.789596           2       True         14
8      WeightedEnsemble_L2  -17.367392  -45.678779        2.994894
113.829946  141.643967                 0.005663                0.000805
0.389164           2       True         10
9   RandomForestMSE_BAG_L1  -18.277222  -54.140131        0.789135
2.616562   16.006378                 0.789135                2.616562
16.006378           1       True          5
10         CatBoost_BAG_L1  -19.258490  -54.972334        0.046041
0.262134  206.944024                 0.046041                0.262134
206.944024           1       True          6
11       LightGBMXT_BAG_L1  -19.335794  -46.273607        0.990943
56.480547   46.595445                 0.990943               56.480547
46.595445           1       True          3
12  NeuralNetFastAI_BAG_L2  -19.646259  -42.846275        4.350618
121.991178  413.195437                 0.192059                1.387108
56.142241           2       True         16
13          XGBoost_BAG_L1  -19.717612  -53.716756        0.270837
0.630693    5.508166                 0.270837                0.630693
5.508166           1       True          9
14         LightGBM_BAG_L1  -19.890861  -48.485993        1.045494
53.506534   45.674402                 1.045494               53.506534
45.674402           1       True          4
15  NeuralNetFastAI_BAG_L1  -20.137444  -60.424840        0.163659
1.225498   32.978578                 0.163659                1.225498
32.978578           1       True          8
16    LightGBMLarge_BAG_L2  -86.764000 -121.458419        4.221751
120.715600  358.779523                 0.063192                0.111530
1.726327           2       True         18
17   KNeighborsUnif_BAG_L1 -161.576210 -210.466014        0.012347
1.640172    0.072643                 0.012347                1.640172
0.072643           1       True          1
18   KNeighborsDist_BAG_L1 -166.746539 -252.720925        0.018066
1.648211    0.070871                 0.018066                1.648211
0.070871           1       True          2
```



[22]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

5 Submit

[23]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
Loaded data from: X_test_raw.csv | Columns = 33 / 33 | Rows = 4608 -> 4608
```

[24]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
↪right_on=["time", "location"], left_on=["ds", "location"])
```

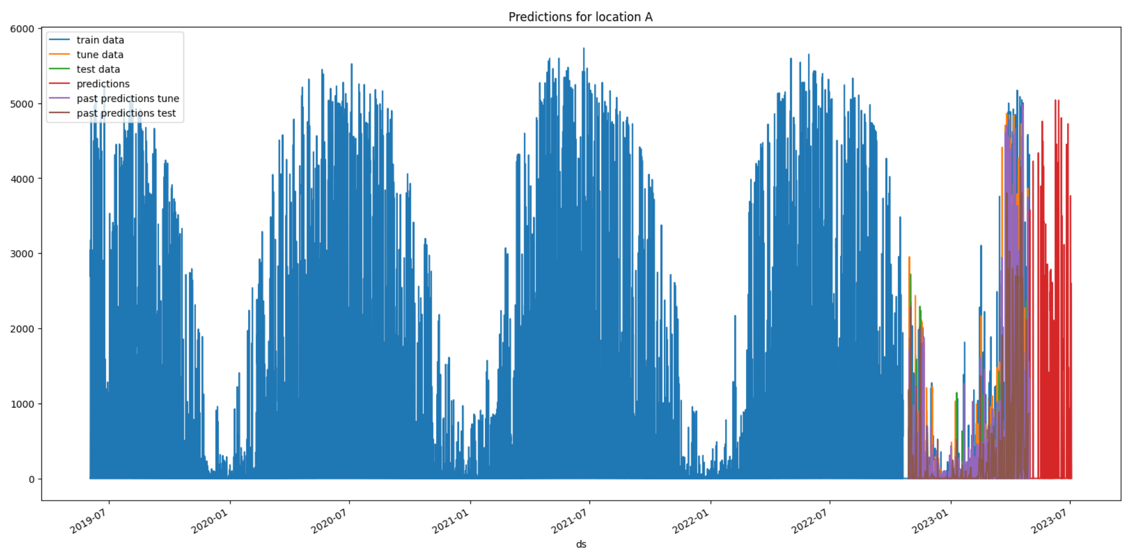
```
#test_data_merged
```

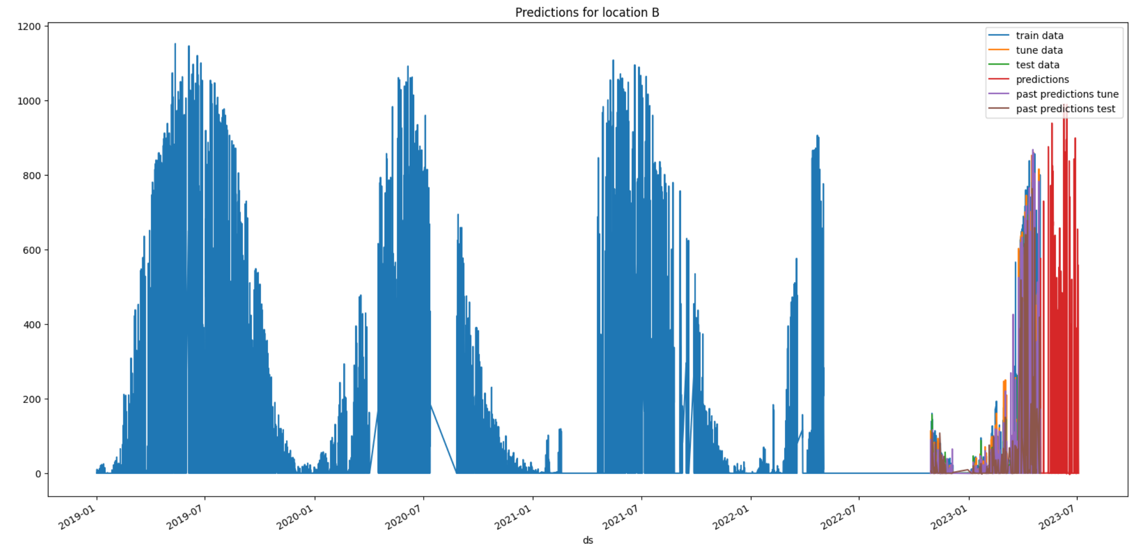
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

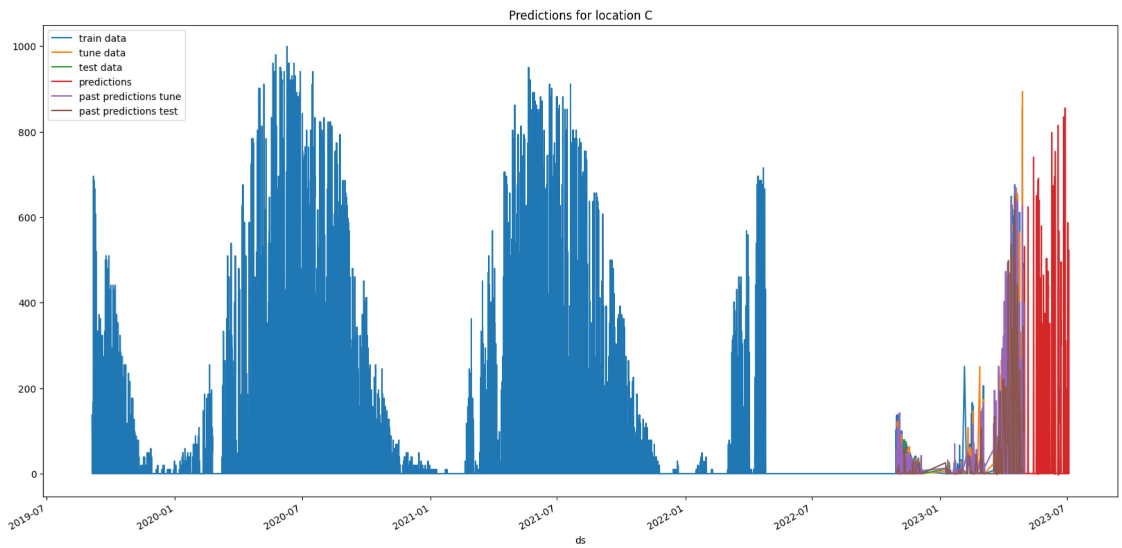
[25]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
 ↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(test_data[test_data["location"] == loc])
```

[26]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
```

```
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions test")


    # title
    ax.set_title(f"Predictions for location {loc}")
```







[27]:
```python
temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())
```

```python
# Instead of clipping, shift all prediction values up by the largest negative␣
↪number.
# This way, the smallest prediction will be 0.
elif shift_predictions:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred["prediction"] = pred["prediction"] - pred["prediction"].min()
        print("Smallest prediction after clipping:", pred["prediction"].min())

elif shift_predictions_by_average_of_negatives_then_clip:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
        # if not nan
        if mean_negative == mean_negative:
            pred["prediction"] = pred["prediction"] - mean_negative

        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -2.8599703
Smallest prediction after clipping: 0.0
Smallest prediction: -3.4934907
Smallest prediction after clipping: 0.0
Smallest prediction: -3.0635557
Smallest prediction after clipping: 0.0
```

[27]:

	id	prediction
0	0	0.167035
1	1	0.251438
2	2	0.335102
3	3	53.541954
4	4	295.727142
..	…	…
715	2155	52.114365
716	2156	23.598484
717	2157	9.234374
718	2158	2.891260

```
719  2159    0.147616

[2160 rows x 2 columns]
```

[28]:
```
# Save the submission DataFrame to submissions folder, create new name based on␣
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_109.csv
jall1a
```

[ ]:
```
# feature importance
# print starting calculating feature importance for location A with big text␣
↪font
print("\033[1m" + "Calculating feature importance for location A..." +␣
↪"\033[0m")
predictors[0].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +␣
↪"\033[0m")
predictors[1].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +␣
↪"\033[0m")
predictors[2].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "C"], time_limit=60*10)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),␣
↪"autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
↪ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
```

```
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

```
[ ]:
```